UNIFACCAMP
MESTRADO EM CIÊNCIA DA COMPUTAÇÃO

Análise de Algoritmos e Complexidade da Computação
Lista de Exercícios 1

Prof. Osvaldo.

1. Descreva alguns problemas associados ao emprego de metodologias de análise de algoritmos baseadas em experimentação?

2. Simule a execução dos seguintes comandos.

a) c := 1;
```
para i := 1 até 5 faça
   c = c + 1;
```

b) c := 1;
```
para i := 1 até 4 faça
   para j := 1 até 3 faça
      c := c + 1;
```

c) c := 1;
```
para i := 1 até 4 faça
   para j := 1 até i faça
      c := c + 1;
```

d)
```
i := 1; c := 1;
enquanto (i ≤ 32)
{ c := c + 1;
  i := i*2;
}
```

e)
```
i := 32; c := 1;
enquanto (i ≥ 1)
{ c := c + 1;
  i := i div 2;  // "div" é divisão inteira.
}
```

f) Observação: neste exercício, assuma que x = 3, n = 5 e A = [5, 2, 1, 3, -1].

```
i := 1; c := 1;
enquanto (i ≤ n e x ≤ A [i])
{ c := c + 1;
  i := i + 1;
}
para j := i até n faça c := c + 1;
```

3. Diga, para cada caso, quantas vezes o comando fictício “S” será executado. Assuma que “S” não altera o valor da variável de controle dos laços.

a)
```
para i := 1 até 2*n faça
    S;
```

b)
```
para i := 1 até n² faça
    S;
```

c)
```
para i := 1 até n faça
    para j := 1 até i faça
       S;
```

d)
```
para i := 1 até 2*n faça
    para j := 1 até i faça
       S;
```

e)
```
para i := 1 até 2ⁿ faça
    para j := 1 até i faça
       S;
```

f)
```
i := 1;
enquanto (i ≤ 2ⁿ)
{ S;
  i := i*2;
}
```

g)
```
i := 2ⁿ;
enquanto (i ≥ 1)
{ S;
  i := i div 2;  // "div" é divisão inteira.
}
```

h)
```
i := 9;
enquanto (i ≤ n)
{ S;
  i := i*3;
}
```

i)

```
i := 1;
enquanto (i ≤ n e x ≤ A [i])
{ S;
  i := i + 1;
}
para j := i até n faça
  S;
```

4. Calcule as complexidades de tempo e de espaço do algoritmo abaixo.

```
Algoritmo Termo3 (n)
Entrada: n, um número natural, isto é, n é inteiro e n
≥ 1.
Saída: t, o n-ésimo termo da seqüência 3, 6, 9, ... .
{
    i := 1; t := 0;

    enquanto ( i ≤ n ) {
        t := t + 3;
        i := i + 1;
     }
}
```

5. O algoritmo abaixo calcula o *n*-ésimo termo da seqüência da Fibomacci (1, 1, 2, 3, 5, 8, 13, ...). Calcule a complexidade de tempo deste algoritmo. Supondo que você possua um computador capaz de realizar 10^6 somas por segundo, quanto tempo este computador levará para calcular o centésimo termo desta seqüência.

```
Algoritmo Fibonacci (n)
Entrada: n, um número natural, isto é, n é inteiro e n
≥ 1.
Saída: t, o n-ésimo termo da seqüência de Fibonacci.
{
    aa := 1; a := 1; f := 1; i := 3;

    enquanto ( i ≤ n ) {
        aux := f; f := a + aa;
        aa := a; a := aux;
        i := i + 1;
     }

    retornar f;
}
```

6. O algoritmo a seguir retorna o maior elemento em de um vetor. Calcule a complexidade de tempo deste algoritmo.

```
Algoritmo Maior (A, n)
Entrada: A, um vetor de n elementos (n ≥ 1).
Saída: o maior elemento do vetor.
{
    Max := A [1];

    para i := 2 até n faça
       se (Max < A [i])
         Max := A [i];

    retornar Max;
}
```

7. O algoritmo a seguir ordena um vetor. Calcule a complexidade de tempo deste algoritmo. O método de ordenação subjacente a este algoritmo é conhecido como método da bolha (*bubble sort*).

```
Algoritmo BubbleSort (A, n)
Entrada: A, um vetor de n elementos (n ≥ 1).
Saída: o vetor A ordenado crescentemente.
{
    para i := 1 até n - 1 faça
       para j := 1 até n - i faça
           se ( A [j] > A [j + 1] )
           {
             t := A [j];
             A [j] := A [j + 1];
             A [j + 1] := t;
           }

    retornar A;
}
```

8. Outro algoritmo de ordenação conhecido como *Selection Sort* é apresentado a seguir. Qual a sua complexidade? O algoritmo `IndiceMenor`, utilizado pelo `SelectionSort`, retorna o índice do menor elemento do vetor X entre as posições i e n, e a sua complexidade de tempo é $T(n) = n - i + 1$.

```
Algoritmo SelectionSort (X, n);
Entrada: vetor X com n elementos (n ≥ 1).
Saída: o vetor X ordenado em ordem crecente.
{
     i :=  1;
     enquanto ( i ≤ n) {
         m := IndiceMenor (X, n, i);
         troca := X [i];
```

```
            X [i] := X [m]; X [m] := troca;
            i := i + 1;
        }
      retornar X;
  }
```

9. O que são as complexidades de pior caso (ou complexidade pessimista), melhor caso (ou complexidade otimista) e de caso médio?

10. Uma modificação do algoritmo de ordenação BubbleSort é apresentada abaixo. Nesta versão caso em uma passada verifica-se que o vetor está ordenado, então o algoritmo pode ser encerrado. Calcule as complexidades de tempo de pior caso, de melhor caso e de caso médio deste algoritmo.

```
Algoritmo BubbleSort (A, n)
Entrada: A, um vetor de n elementos (n ≥ 1).
Saída: o vetor A ordenado crescentemente.
{
    troca := verdadeiro; i := 1
    enquanto (i ≤ n e troca = verdadeiro)
    {
       troca := falso;
       j := 1;
       enquanto (j ≤ n – i)
       {
           se ( A [j] > A [j + 1] )
           {
             t := A [j];
             A [j] := A [j + 1];
             A [j + 1] := t;
             Troca := verdadeiro;
           }
           j := j + 1;
       }
       i := i + 1;
    }

    retornar A;
 }
```

11. Esboce os gráficos das seguintes funções: log (n), n, n log (n), n^2, 2^n.

12. Calcule o tempo em segundos, minutos, dias ou anos que gastaria um processador capaz de realizar 10^6 operações por segundo para executar uma instância de tamanho n = 100 de um algoritmo cuja complexidade é definida pelas funções: log n, $\sqrt{n}$, n, n log n, n^2, n^3, 2^n, $n!$.

13. Ordene a lista de funções a seguir usando a notação O. Agrupe as funções que são Θ uma da outra.

$6n \log n$	2^{100}	$\log \log n$	$\log^2 n$	$2^{\log n}$
2^{2n}	$\sqrt{n}$	$n^{0.01}$	$1/n$	$4n^{3/2}$
$3n^{0.5}$	$5n$	$2n \log^2 n$	2^n	$n \log_4 n$
4^n	n^3	$n^2 \log n$	$4^{\log n}$	$\sqrt{\log n}$

14. Mostre que:
    a) $n^2 = O(n^3)$.
    c) $2^{n-1} = \Theta(2^n)$.
    d) $\log^a n = O(n^b)$, para $a, b > 0$.

15. Mostre que $\log_b (f(n)) = \Theta(\log (f(n))$, para toda constante $b > 1$.

16. Mostre que se $f(n) = O(g(n))$ e $g(n) = O(h(n))$ então $f(n) = O(h(n))$.

17. Mostre que se $f(n) = O(s(n))$ e $g(n) = O(r(n))$ então $f(n) + g(n) = O(s(n) + r(n))$.

18. Ache um contra-exemplo para a afirmação: se $f(n) = O(s(n))$ e $g(n) = O(r(n))$ então $f(n) - g(n) = O(s(n) - r(n))$.

19. Alberto e Bernardo estão discutindo a performance de seus algoritmos de ordenação. Alberto afirma que o seu algoritmo $O(n \log n)$ é sempre mais rápido do que o algoritmo $O(n^2)$ de Bernardo. Para decidir, eles implementam em computador e executam os dois algoritmos em uma série de dados de teste. Para o espanto de Alberto, eles descobrem que quando $n < 100$ o algoritmo $O(n^2)$ é mais rápido e apenas quando $n \geq 100$ é que o algoritmo $O(n \log n)$ é mais rápido. Explique como esta situação pode ocorrer. Se for preciso, forneça exemplos numéricos.

20. A segurança de comunicações é extremamente importante em redes de computadores e, uma forma típica de garanti-la é *criptografar* as mensagens. Sistemas criptográficos típicos para a transmissão segura de dados através de redes são baseados no fato de que não são conhecidos algoritmos eficientes para a fatoração de números inteiros muito grandes. Assim, se podemos representar uma mensagem secreta como um grande número primo p, podemos transmitir pela rede o número $r = p \cdot q$, onde $q > p$ é outro grande número primo que serve como chave de *encriptação*. Um espião que obtenha o número r transmitido teria de fatorá-lo para descobrir a mensagem secreta p.

    Usar fatoração para descobrir a mensagem é bastante difícil sem que se conheça a chave q. Para entender por que, considere o seguinte algoritmo simples de decifração:

Para cada inteiro tal que $1 < p < r$, verifique se p divide r. Se dividir, imprima que " A mensagem secreta é p" e, caso contrário, continue.

a) Suponha que um espião use esse algoritmo em um computador que possa realizar em 1 microssegundo (10^{-6} segundo) a operação de divisão entre dois inteiros de 100 bits cada um. Forneça uma estimativa do tempo necessário, no pior caso, para decifrar a mensagem secreta se r possui 100 bits.

b) Qual a complexidade do algoritmo acima? Dica: a entrada do algoritmo é um grande número r, cujo tamanho n é o número de bytes necessários para armazená-lo. Ou seja, em bytes, $n = \log_2 (r) / 8$. Suponha que cada divisão pode ser realizada em um tempo proporcional ao tamanho de r.

21. Dê um exemplo de uma função positiva $f(n)$ de tal forma que $f(n)$ não seja nem $O(n)$, nem $\Omega(n)$.

22. Prove por indução que a soma das $n + 1$ primeiras potências inteiras de 2, isto é, $1 + 2 + 4 + ... + 2^n$, é $S(n) = 2^{n+1} - 1$.

23. Prove por indução que a soma $1 + 3 + 5 + ... + 2n - 1 = n^2$, i.e., a soma de números ímpares sempre é um quadrado perfeito.

24. Prove que se n é um número natural e $1 + x > 0$, então $(1 + x)^n \geq 1 + nx$.

25. Prove que a quantidade de nós de uma árvore binária cheia é igual a $2^{h+1} - 1$, onde h é a altura da árvore. Uma árvore binária cheia é uma árvore para a qual, se v é um nó de alguma de suas subárvores vazias, então v se localiza no último nível.

26. Ache uma expressão para a seguinte soma e prove a sua afirmação.

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^n}$$

27. Prove que $x^n - y^n$ é divisível por $x - y$ para todos os números naturais x, y ($x \neq$ y) e n.

28. Prove a seguinte inequação:

$$\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^n} < 1$$

para todo $n \geq 1$.